VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
Fakulta strojního inženýrství
Ústav automatizace a informatiky

Ing. Petr Krček

PLÁNOVÁNÍ CESTY AUTONOMNÍHO LOKOMOČNÍHO ROBOTU NA ZÁKLADĚ STROJOVÉHO UČENÍ

AUTONOMOUS LOCOMOTIVE ROBOT PATH PLANNING ON THE BASIS OF MACHINE LEARNING

Zkrácená verze Ph.D. Thesis

Obor: Inženýrská mechanika
Školitel: RNDr. Jiří Dvořák, CSc.

KLÍČOVÁ SLOVA

Plánování cesty, neholonomní robot, případové usuzování, genetické algoritmy.

KEYWORDS

Path planning, nonholonomic robot, case-based reasoning, genetic algorithms.

OBSAH

1 ÚVOD

Potřeba použití robotů v nestálých prostředích vedla k vytvoření *autonomního robotu*. Autonomní robot je takový stroj, který je za použití metod umělé inteligence schopen vykonávat úkoly zcela samostatně. Takový robot musí sám optimálně reagovat na dynamické změny v prostoru. Většina autonomních mobilních (lokomočních) robotů používá při navigaci operace nad mapou prostředí. V této souvislosti se hovoří o tzv. *mapově orientované navigaci*.

Obecně je mapově orientovaná navigace dělena do tří procesů (Meyer 2003): lokalizace, učení mapy a plánování cesty. *Lokalizace* je proces odvození aktuální pozice robotu. *Učení mapy* je proces pamatování dat, která jsou získávána během provozu robotu. *Plánování cesty* je proces výběru dílčích kroků k dosažení cíle z aktuální pozice tak, aby v průběhu plnění plánu nenastala kolize s překážkami v pracovním prostoru. Lokalizace a učení mapy jsou vzájemně závislé procesy. Používání mapy pro lokalizaci robotu vyžaduje existující mapu, budování mapy vyžaduje pozici k odhadování vztahu k neúplné doposud naučené mapě. Plánování cesty je naopak dosti nezávislý proces, který je spuštěn, až když mapa existuje a je odhadnuta pozice robotu.

Plánování cesty v dynamickém částečně známém či neznámém prostředí je obtížným problémem. Schopnost robotu přizpůsobovat svoje chování změnám prostředí může být zajištěna pomocí metod strojového učení. V souvislosti s plánováním cesty se z metod strojového učení uplatňují především případové usuzování, neuronové sítě, posilované učení, ale i genetické algoritmy.

Výsledkem disertační práce je inteligentní systém plánování cesty respektující kinematická omezení robotu. Jako základní stavební kámen této práce byl zvolen systém případového usuzování. Systém byl navržen tak, aby nebyl vázán na jeden kinematický model robotu. Funkčnost systému je podložena experimenty.

2 PLÁNOVÁNÍ CESTY

Když má robot použitelnou mapu, má k dispozici odhad své pozice uvnitř této mapy a je zadána cílová pozice uvnitř této mapy, potom by robot měl být schopen pohybu z jeho aktuální pozice do cílové pozice. Plánování cesty spočívá v nalezení postupu, jak má robot dosáhnout cíle. S přehledem metod pro plánování cesty robotu seznamují např. práce (LaValle 2006) nebo (Meyer 2003).

Při plánování (LaValle 2006) se každá situace robotu nazývá *stav* a označuje se jako x. Množina všech možných stavů se nazývá *stavový prostor* označovaný symbolem X. Aplikujeme-li na aktuální stav x akci u, tak se aktuální stav změní na stav x', který je daný přechodovou funkcí $x' = f(x, u)$. Akce, které mohou být aplikovány na stav x, tvoří *akční prostor* stavu x označovaný jako $U(x)$. Všechny možné akce všech stavů tvoří množinu $U = \bigcup_{x \in X} U(x)$. Množina $X_G \subset X$ obsahuje přípustné cílové stavy. Úkolem plánování je nalézt konečnou posloupnost akcí,

jejichž aplikováním postupně transformujeme počáteční stav x_1 do některého stavu z X_G.

Mapy robotů rozlišujeme na *metrické* a *topologické*. V metrických mapách jsou v běžné soustavě souřadnic uloženy pozice různých objektů (především překážek), se kterými se může robot setkat. Naproti tomu v topologické mapě jsou uloženy oblasti, kterých robot může dosáhnout. Spolu s těmito oblastmi jsou uloženy informace o jejich vzájemné pozici.

Plánování cesty přímo nad metrickou mapou není z hlediska časové náročnosti efektivní, neboť odpovídající stavový prostor X je nespočetně nekonečný. Časovou náročnost lze snížit vyhledáváním cesty v topologické mapě. Topologickou mapu obdržíme z metrické mapy tzv. diskretizací prostoru. Většina plánovacích metod tedy začíná diskretizací a potom používá některou z metod hledání v topologické mapě. Hovoříme potom o diskrétním plánování. Po provedení diskretizace je prostor X spočetný a často i konečný.

Nejsou-li na robot při plánování kladena dynamická diferenciální omezení (viz podkapitola 2.3), pak se někdy hovoří o stavovém prostoru X jako o konfiguračním prostoru C a o stavu x jako o konfiguraci q, platí tedy rovnost $X = C$. Počet rozměrů konfiguračního prostoru často odpovídá počtu stupňů volnosti robotu.

V souvislosti s překážkami se zavádí pojem volného konfiguračního prostoru C_{free}, který je podmnožinou konfiguračního prostoru a obsahuje pouze takové konfigurace robotu, které nejsou v kolizi s žádnou překážkou.

Pro základní (holonomní) plánování cesty platí, že cesta může být snadno určena mezi dvěmi konfiguracemi v C přímou čarou, nejsou-li konfigurace na příslušné úsečce v kolizi s překážkou. Omezení kladená na robot jsou dána pouze množinou povolených konfigurací v C_{free}. Tato omezení se označují jako globální.

2.1 ZÁKLADNÍ TYPY DISKRETIZACE PROSTORU

První třída metod pro diskretizaci extrahuje topologickou mapu z metrické mapy dekompozicí volného konfiguračního prostoru C_{free} do buněk korespondujících s uzly v topologické mapě (Meyer 2003). Do této třídy metod patří typicky *rastrové metody*. Topologická mapa se v tomto případě dostane pomocí proložení mřížky přes metrickou mapu. Jiné metody konstruují topologickou mapu na základě dekompozice volného prostoru do konvexních polygonů. Mezi tyto metody patří např. *vertikální buňková dekompozice* a metody založené na *triangulaci* (LaValle 2006).

Druhá třída metod pro diskretizaci provádí rozklad volného konfiguračního prostoru C_{free} do dílčích přípustných cest, které propojují klíčové body rozptýlené v tomto prostoru (Meyer 2003). Takto zkonstruovaný graf se nazývá *mapa cest*. V základních variantách metod této třídy je získaná mapa cest vhodná pro bodový robot. Chceme-li plánovat cestu pro nebodový robot, je nutné provést úpravu metrické mapy tzv. *zvětšením překážek*.

Mapa cest může být odvozena například z *grafu viditelnosti* (Priya 2006), který spojuje přímkami takové vrcholy překážek, které jsou navzájem viditelné. V tomto případě klíčové body mapy cest jsou vrcholy překážek, které umožňují výpočet nejkratších cest ležících blízko překážek. Metody pro výpočet mapy cest, které maximalizují vzdálenost od překážek, využívají *Voronoiův diagram* (Šeda 2005). V tomto případě klíčové body mapy cest jsou body ekvidistantní nejméně třem překážkám.

Dalším případem metod generujících mapu cest jsou *metody založené na vzorkování*. Princip těchto metod spočívá v prohledávání nespočetně nekonečného konfiguračního prostoru *C* podle daného typu vzorkování. Algoritmus vzorkování je však ukončen po dosažení daného počtu vzorků, tedy přípustných konfigurací robotu v metrické mapě. Vzorky představují uzly grafu, který tvoří topologickou mapu. Vzhledem k tomu, že tyto metody jsou většinou založeny na náhodném vzorkování, tak nezaručují úplnost vzorků a tudíž ani nalezení cesty.

Na *Inkrementálním vzorkování a prohledávání* jsou založeny metody z rodiny *rychle mapujících hustých stromů (Rapidly exploring Dense Trees – RDTs)*. U těchto metod se topologická mapa vytváří při každém hledání cesty jako strom, který pozvolna zvyšuje pokrytí konfiguračního prostoru. Metody typu RDTs končí s generováním stromu v okamžiku nalezení řešení. Pokud jsou přidávány konfigurace do stromu náhodně, pak se jedná o metodu *rychle mapujících náhodných stromů (Rapidly exploring Random Trees – RRTs)*. Detailně popisuje tuto metodu např. článek (LaValle 2001). Byly navrženy také různé modifikace tohoto algoritmu (např. Krejsa 2005).

2.2 METODY PLÁNOVÁNÍ CESTY

Pro diskrétní plánování cesty v topologické mapě lze použít klasických přístupů prohledávání grafu nebo lze použít některou z metod strojového učení, která poskytuje pro danou aplikaci rychlejší výpočet anebo vhodnější řešení.

2.2.1 Klasické přístupy

Mezi skupinu klasických algoritmů pro prohledávání grafu patří především *Dijkstrův algoritmus*, *A** a jeho varianty. Pro přijatelné velikosti map obvykle používané v robotice poskytují tyto algoritmy účelné použití. Pokud je mapa příliš velká, je navíc možné užít *dynamické programování*, využívající Bellmanův princip optimality (Meyer 2003). Mezi méně známé varianty algoritmu A* patří algoritmy D* a ARA*. Algoritmus D* je schopen se vyrovnat se změnou v grafu v průběhu již spouštěného prohledávání, takže je předurčen pro použití v částečně známých prostředích. Algoritmus ARA* lze zařadit mezi *anytime* algoritmy. Tyto algoritmy jsou schopny poskytnout výsledek po velmi krátké době výpočtu, což je vhodné pro aplikace pracující v reálném čase. Výsledek však nelze považovat za nejlepší řešení. V průběhu výpočtu se výsledek zpřesňuje. V práci (Ferguson 2005) je kromě algoritmů D* a ARA* představena také metoda oba přístupy kombinující.

2.2.2 Přístupy založené na strojovém učení

Plánování cesty pomocí neuronové sítě vychází často z Hopfieldovy neuronové sítě, a to z její spojité realizace (např. Ritthipravat 2002). Lze se však setkat i s návrhy modifikovaných či vlastních sítí (např. Lebedev 2005). Jako výhoda použití neuronové sítě je obvykle uváděna schopnost plánování v nestálém prostředí.

Genetické algoritmy (GA) jsou schopné pokrýt rozsáhlý prostor prohledávání při relativně malých nárocích na výpočetní zdroje. Schopnost adaptace nalezeného řešení na spojitě se měnící dynamické prostředí a generovat řešení v reálném čase demonstrují např. práce (Burchardt 2006) a (Zheng 2004). Nedostatkem GA je, že nezaručují nalezení globálně optimálního řešení.

Pro plánování cesty pomocí rojové inteligence jsou nejčastěji používány *metoda rojení částic* (např. Lei 2006) a *metoda mravenčí kolonie* (např. Viet 2008). Obecně je výhodou těchto metod schopnost přizpůsobit se dynamickým změnám v prostoru.

Strojové učení může být také realizováno použitím *případového usuzování* (*Case Based Reasoning – CBR*). CBR řeší nový problém adaptací známých řešení podobných problémů, které byly již řešeny v minulosti. Zdá se, že CBR je pro navigaci robotu vhodnou metodou, neboť v řadě aplikací robot řeší často podobné úkoly. Hlavní cyklus CBR může být popsán čtyřmi základními kroky (Aamodt 1994): (i) nalezení nejvíce podobného případu či případů; (ii) použití informací a znalostí z tohoto případu (případů) k řešení daného problému; (iii) kontrola navrženého řešení; (iv) uložení vhodných částí tohoto řešení k opětnému použití při řešení budoucích podobných problémů. V situacích, kdy systém případového usuzování začíná pracovat s prázdnou bází případů, nebo když získané řešení není dost dobré, je nezbytné použít některou další metodu hledání cest. Případy mohou reprezentovat kompletní cesty (Kruusmaa 2003) nebo mohou být rozloženy do několika případů a uchovávány např. v *případovém grafu* (Haigh 1994).

Posilované učení lze použít přímo pro plánování cesty (Drummond 2002). Existují však také práce, ve kterých je posilované učení použito jako prostředek pro učení vah neuronové sítě, nebo naopak jsou pomocí genetických algoritmů nastavovány parametry algoritmu Q-učení.

2.3 NEHOLONOMNÍ PLÁNOVÁNÍ CESTY

O *neholonomním* plánování cesty se hovoří tehdy, jsou-li na robot kladena diferenciální omezení, která nejsou plně integrovatelná, tzn., že nemohou být převedena do omezení nezahrnujících derivaci (LaValle 2006). Neholonomní plánování zahrnuje jak plánování s kinematickými, tak s dynamickými omezeními. Pro častější kinematické plánování platí, že $X = C$, což pro dynamické plánování již neplatí. Stavový prostor X zde bývá tvořen fázovým prostorem konfiguračního prostoru C a potom tedy stav $x = (q, \dot{q})$ pro $q \in C$.

Diferenciální omezení udávají povolenou rychlost v každém stavu. Takto představují *lokální* omezení, která tvoří doplněk ke globálním omezením. Základní

metody plánování uvedené v podkapitole 2.2 ignorují tato lokální omezení a předpokládají, že diferenciální omezení budou řešena až v průběhu provádění plánu.

Diferenciální omezení je často vyjádřeno jako $\dot{x} = \varphi(x, u)$ na spojitém stavovém prostoru *X*. Jedná se vlastně o v čase spojitý protějšek k přechodové funkci $x' = f(x, u)$ (LaValle 2006). Spojitá přechodová funkce $\varphi(x, u)$ udává místo nového stavu robotu jeho rychlost ve stavu *x*. Budoucí stav, který splňuje diferenciální omezení, je získán integrací rychlosti. Výslednou cestu charakterizuje *akční trajektorie* $\tilde{u} : \langle 0, \infty) \to U$, kde U je akční prostor $U \subseteq R^m$. Akce v konkrétním čase *t* je vyjádřena jako $\tilde{u}(t)$. Z akční trajektorie lze odvodit s*tavovou trajektorii* $\tilde{x}$ pomocí integrace

$$\tilde{x}(t) = \tilde{x}(0) + \int_0^t \varphi(\tilde{x}(t'), \tilde{u}(t'))dt', \qquad (1)$$

která integruje $\dot{x} = \varphi(x, u)$ z počátečního stavu $\tilde{x}(0)$.

Prostor stavů robotu je obvykle charakterizován orientovaným grafem $G = (V, E)$, kde V je množina uzlů odpovídajících stavům robotu a E je množina hran reprezentujících akční trajektorie $e : \langle 0, \Delta t \rangle \to U$. Obvykle každé hraně grafu odpovídá jedna akce. Počáteční vrchol hrany reprezentuje výchozí stav a koncový vrchol hrany odpovídá stavu, který získáme aplikací této akce na výchozí stav po dobu Δt. Jestliže tedy máme nějakou hranu *e* a jí odpovídající akci *u*, pak platí $e(t) = u$ pro všechna $t \in \langle 0, \Delta t \rangle$. Stavovou trajektorii hrany $\tilde{x}_e$ získáme pomocí výše uvedené integrace (1), kde $\tilde{u}(t) = e(t)$.

Neholonomní plánovací metody vycházejí z metod založených na vzorkování. Obecně se dají tyto metody popsat následujícími kroky (LaValle 2006):

1. **Inicializace:** Nechť $G = (V, E)$ representuje orientovaný graf, kde V obsahuje počáteční stav x_I a další možné stavy v X_{free} a kde E je prázdná množina hran.
2. **Výběr uzlu pro expanzi:** Vybereme stav $x_{cur} \in \bigcup_{e \in E} \bigcup_{t \in \langle 0, \Delta t \rangle} x_e(t)$. Pokud je množina E prázdná, položíme $x_{cur} = x_I$.
3. **Lokální plánování:** Generujeme akci *u* takovou, aby $\tilde{x}(0) = x_{cur}$ a $\tilde{x}(\Delta t) = x_r$ pro nějaké $x_r \in X_{free}$, které může nebo nemusí být vrcholem grafu *G*. Vzniklá stavová trajektorie musí obsahovat uzly, které náleží X_{free}. Není-li tomu tak, opakujeme krok 2.
4. **Vložení hrany do grafu:** Akci *u* reprezentující akční trajektorii *e* vložíme do *E*. Pokud $x_{cur} \notin V$ nebo $x_r \notin V$, pak je do *V* přidáme. Pokud x_{cur} obsahuje stavová trajektorie $\tilde{x}_e$ některé hrany $e \in E$, pak je nutno hranu *e* rozdělit podle uzlu x_{cur}.
5. **Kontrola řešení:** Ověříme, zda graf *G* neobsahuje hledanou cestu. Pokud ano, algoritmus končí úspěšně. V některých případech je možné ve výsledné stavové trajektorii tolerovat malou mezeru.
6. **Návrat zpět na krok 2:** Je-li splněna nějaká jiná ukončovací podmínka, než nalezení řešení, algoritmus končí neúspěchem. Jinak se vrátíme na krok 2.

Výběr uzlů x_{cur} a x_r již záleží na konkrétní metodě. LaValle (2006) uvádí tři takové přístupy: *začlenění diskretizace stavového prostoru*, metody *RDT* a *prohledávání na mřížce*, které je však především určeno pro roboty s dynamickými omezeními. Existují také přístupy, které jsou založeny např. na GA či rojení částic.

3 NÁVRHY METOD PLÁNOVÁNÍ CESTY

3.1 PLÁNOVÁNÍ CESTY NA MŘÍŽCE

Navržené metody z této oblasti jsou založeny především na případovém usuzování. První návrhy se snaží zlepšit metodu autorů Kruusmaa & Svensson (2003), ve které je stupeň využití minulých zkušeností značně redukován, protože jsou pro znovupoužití uvažovány jen kompletní cesty. Tyto nevýhody jsou odstraněny v práci (Krček, Dvořák & Hodál 2005). Je zde navržen algoritmus, který pracuje nad případovým grafem a pro dohledávání neznámých částí cest používá hledání pomocí genetických algoritmů. Při ukládání cesty do báze případů se tato cesta uchovává jako jedna nebo více hran případového grafu, neboť je nutné dodržet podmínku, že se uložené případy smí protínat pouze v krajních bodech. Z provedených experimentů vyplývá, že použití případového grafu pro velké mřížky přináší významnou úsporu ve srovnání se samotným genetickým algoritmem a s použitím celých cest jako případů.

Na tuto práci navazují práce (Hodál, Dvořák & Krček 2005), kde je používáno heuristické lokální hledání a (Dvořák & Krček 2005), kde je CBR kombinováno s grafovými algoritmy. V této práci je algoritmus pracující nad případovým grafem zjednodušen zjemněním struktury případového grafu. Metoda užívající tento algoritmus používá jako pomocnou metodu A*.

3.2 KOMBINACE PŘÍPADOVÉHO GRAFU A RRT

Po prvních pokusech s algoritmem RRT byl v práci (Krček & Dvořák 2006) představen přístup pro plánování cesty robotu typu „neholonomní tříkolka s diferenciálním řízením“, který spočívá v kombinaci případového usuzování s RRTs. Algoritmus RRT bylo nutné pro efektivní spojení s metodou případového usuzování modifikovat pro vyhledávání do více cílů.

Báze případů je organizována jako případový graf, přičemž každý případ je reprezentován dvojicí opačně orientovaných hran. Při ukládání úspěšně absolvované cesty robotu do báze případů se jako samostatný případ ukládá každá triviální hrana cesty nalezená algoritmem RRT. K uložení jednotlivých hran ovšem může dojít pouze tehdy, pokud se v bázi případů ještě nevyskytují podobné případy. Uzly případového grafu odpovídají konfiguracím robotu ohraničujícím jednotlivé triviální hrany a jsou ohodnoceny souřadnicemi středu robotu. Pokud hrana v případovém grafu protíná jinou hranu, potom se snažíme propojit tyto hrany (pokud je to možné) doplněním dalších uzlů a triviálních hran.

Kombinovaný algoritmus CBR a RRT se pokouší spojit jak počáteční, tak i cílovou konfiguraci robotu s nejbližšími uzly případového grafu (k tomu se používá algoritmus RRT s malým počtem iterací). V takto vzniklém grafu se pak hledá optimální cesta pomocí upraveného Dijkstrova algoritmu. Tato úprava zohledňuje to, že v důsledku kinematických omezení robotu možnost pokračování z aktuálního uzlu závisí na hraně, po které se robot do tohoto uzlu dostal. V případě neúspěchu kombinovaného algoritmu se cesta z počáteční do cílové konfigurace robotu hledá pomocí algoritmu RRT s vyšším počtem iterací.

3.3 GENETICKÝ ALGORITMUS

Již v práci (Krček & Dvořák 2007) je navržen genetický algoritmus, u nějž byla zkoumána možnost použití části „naučené“ populace z nějakého předchozího řešení jako část počáteční populace nové úlohy. Při znovupoužití vhodné části „naučené“ populace se tato adaptace ukázala být účelná především u úloh, kde je nutné aktuální řešení přeplánovat v důsledku výskytu náhodné překážky. V práci (Dvořák & Krček 2008) je tento genetický algoritmus vylepšen o další specifické operátory.

Plánování cesty probíhá ve dvourozměrném spojitém prostoru, v němž se vyskytují polygonální překážky popsané svými vrcholy. Dále je předpokládán holonomní robot, který je pro jednoduchost reprezentován jako bod v prostoru. Pro uvažování rozměrů holonomního robotu je nutné zvětšit překážky tak, aby se robot mohl bezpečně pohybovat po hranách těchto zvetšených překážek.

Chromozom vyjadřuje cestu v prostoru a jeho geny představují uzlové body, ve kterých cesta mění svůj směr. Každý gen je přitom tvořen dvojicí souřadnic x a y. U všech chromozomů platí, že počáteční gen reprezentuje pozici startu a poslední gen pozici cíle. Cesta je tudíž tvořena úsečkami spojujícími sousedící geny chromozomu. Chromozom má proměnlivou délku v závislosti na složitosti cesty.

Počáteční populace je získána náhodným generováním, avšak do chromozomů jsou generovány pouze takové uzly, které leží mimo překážky. Počáteční délka každého chromozomu je rovna součtu dané minimální délky a náhodně zvoleného čísla z daného rozsahu. Velikost počáteční populace je dána a zůstává stejná i v následujících generacích.

Fitness funkce je definována dvěma způsoby: pro nepřípustné cesty a pro přípustné cesty. Fitness funkce pro nepřípustnou cestu je dána vztahem:

$$F(P) = f_{collisions}(P) + 1 \tag{2}$$

kde $f_{collisions}(P)$ je počet kolizí cesty P s překážkami. Fitness funkce přípustné cesty P zohledňuje její délku, počet genů a její hladkost takto:

$$F(P) = w_1 f_{length}(P) + w_2 f_{nodes}(P) + w_3 f_{smooth}(P) \tag{3}$$

kde w_k jsou nezáporné normalizované váhy (jejich součet je roven 1),

$$f_{length}(P) = 1 - \frac{d(s,g)}{\sum_{j=1}^{n-1} d(j, j+1)},\ f_{nodes}(P) = 1 - \frac{2}{n},\ f_{smooth}(P) = 1 - \frac{\sum_{j=2}^{n-1} \alpha(j)}{(n-2)\pi}, \quad (4)$$

kde $d(s,g)$ je Euklidovská vzdálenost mezi startem a cílem, $d(j, j+1)$ je délka segmentu mezi uzly j a $j + 1$, n je počet uzlů cesty a $\alpha(j)$ je úhel mezi segmenty ($j - 1, j$) a ($j, j + 1$).

Dílčí kritéria jsou normalizována tak, že jejich nejlepší (ideální) hodnota je rovna 0 a horní mez jejich hodnot je rovna 1. Protože i váhy w_k jsou normalizované, hodnoty fitness funkce (3) leží v intervalu $\langle 0; 1 \rangle$ s nejlepší hodnotou rovnou 0. Minimální hodnota vztahu (2) koresponduje s horní mezí tohoto intervalu. Cílem plánování cesty je nalézt přípustnou cestu minimalizací fitness funkce (3). Díky fitness funkci (2) zrychlíme dobu výpočtu, neboť pro nepřípustné cesty není nutné počítat hodnoty dílčích kritérií.

Dále je navržen následující systém operátorů:

- Operátor *křížení* představuje variantu jednobodového křížení. Nejprve se určí všechny geometrické průsečíky cest dvou vybraných chromozomů. Náhodně zvolený průsečík slouží jako dělicí místo rodičovských chromozomů a vkládá se jako nový gen do potomků mezi části získané z rodičovských chromozomů. Jestliže je počet průsečíků roven nule, pak se dělicí místo v obou rodičovských chromozomech určí jako náhodně vybraný gen.
- Operátor *mutace* náhodně vybere jeden gen z daného chromozomu a náhodně změní jeho souřadnice. Změněný gen musí ležet mimo uvažovanou cestu a mimo překážky. Jsou navrženy dva druhy mutace. První z nich (*velká mutace*) může být aplikována na přípustné i nepřípustné cesty a vzdálenost nové polohy genu od minulé není omezena. Druhá (*malá mutace*) je aplikována jen na přípustné cesty a provádí jemnou změnu souřadnic náhodně vybraného genu tak, aby příslušné segmenty byly nekolizní.
- *Opravný* operátor je aplikován na přípustné chromozomy. Náhodně vybraný kolizní segment (křížící překážku) je vyjmut a na jeho místo je vložena cesta obcházející tuto překážku (podél hran zvětšené překážky).
- Operátor *výměny* může být aplikován na přípustné i nepřípustné chromozomy. Eliminuje dvě následné ostré zatáčky zaměněním souřadnic vybraných sousedících genů.
- Operátor *vyhlazení* je aplikován jen na přípustné chromozomy a udržuje jejich přípustnost. Tento operátor vyhlazuje cestu seříznutím ostré zatáčky. Gen příslušný ostré zatáčce je odstraněn a na jeho místo jsou vloženy dva geny ležící na segmentech vycházejících z odstraněného genu.
- Operátor *zkrácení* je aplikován jen na přípustné chromozomy a udržuje jejich přípustnost. Pokud je to možné, operátor vyjme část cesty mezi dvěma vybranými uzly.

- Operátor *odstranění* je použit v opravném operátoru a v operátorech vyhlazení a zkrácení. Může být však aplikován odděleně na přípustné cesty na základě nějakých heuristických znalostí. Např. je možné odstranit uzel, který téměř nemění směr cesty nebo uzly, které jsou příliš blízko sousednímu uzlu.
- Operátor *zlepšení* systematicky aplikuje na přípustné chromozomy operátory zkrácení a vyhlazení.

Změna populace je prováděna inkrementálně. Prostřednictvím binární turnajové selekce je pro křížení vybrán daný počet párů. Po křížení jsou na získané potomky aplikovány zbývající operátory. Tito potomci jsou přidáni do staré populace a poté se provede seřazení jedinců v populaci podle hodnoty fitness. Pro získání nové populace se provede odstranění chromozomů s nejnižší hodnotou fitness tak, aby velikost populace odpovídala velikosti počáteční populace. Tím je zajištěno, že nejlepší chromozomy staré populace postoupí do populace nové.

Kritérium ukončení algoritmu může být určeno daným počtem generací nebo podmínkou, že nejlepší hodnota fitness v generacích se již nemění více než o malou konstantní hodnotu.

3.4 OPRAVNÝ ALGORITMUS

Cesty navržené metodou popsanou v předcházející podkapitole nelze použít pro neholonomní roboty. Proto je v pracích (Krček & Dvořák 2009) a (Dvořák & Krček 2009) navržen opravný algoritmus, který diskretizuje konfigurační prostor a modifikuje cestu nalezenou pro holonomní robot pomocí heuristické funkce. Tento algoritmus vytváří stromový graf, kde uzly reprezentují konfiguraci robotu a hrany reprezentují odpovídající akce mezi uzly. Kořen stromového grafu odpovídá počáteční konfiguraci robotu. Expandován je vždy uzel s nejlepší hodnotou heuristické funkce.

3.5 GA PRO NEHOLONOMNÍ ROBOT

Problém při transformaci cesty nalezené pro holonomní robot může nastat tehdy, když není natočení robotu ve startu nebo cíli blízké natočení sousední hraně. Pokud v takovéto situaci není robot v prostředí schopen snadného otočení, je nutné, aby s natočením robotu ve startu a cíli počítal již GA. Podobný problém může nastat, když hrany cesty spolu svírají ostrý úhel anebo jsou hrany cesty příliš krátké. Pro řešení těchto problémů je navržena úprava GA z podkapitoly 3.3.

První změna se týká fitness funkcí (2) a (3). Nová fitness funkce pro nepřípustnou cestu je dána vztahem:

$$F(P) = f_{collisions}(P) + f_{ShortSegment}(P) + f_{ColSegment}(P) + f_{ColStart}(P) + f_{ColGoal}(P) + 1 \quad (5)$$

kde $f_{collisions}(P)$ je počet kolizí cesty P s překážkami, $f_{ShortSegment}(P)$ je počet hran cesty P kratších než stanovená mez d_{Min} a pro zbývající funkce platí:

$$f_{ColSegment}(P) = \sum_{i=1}^{n-1} \begin{cases} 1 & \text{pokud } |\beta_i - \beta_{i+1}| > \beta_{SegMax} \\ 0 & \text{jinak} \end{cases}, \tag{6}$$

$$f_{ColStart}(P) = \begin{cases} 1 & \text{pokud } |\beta_1 - \beta_S| > \beta_{StartMax} \\ 0 & \text{jinak} \end{cases}, \tag{7}$$

$$f_{ColGoal}(P) = \begin{cases} 1 & \text{pokud } |\beta_n - \beta_G| > \beta_{GoalMax} \\ 0 & \text{jinak} \end{cases}, \tag{8}$$

kde β_i je natočení *i*-té hrany cesty, β_S (β_G) je natočení robotu ve startu (cíli), β_{SegMax} je maximální povolený úhel mezi každými dvěma hranami a $\beta_{StartMax}$ ($\beta_{GoalMax}$) je maximální povolený úhel mezi natočením robotu ve startu (cíli) a sousedící hranou. Ve fitness funkci pro přípustné cesty P je zavedena nová složka $f_{AngleStartGoal}(P)$, která rozšiřuje funkci (3) takto:

$$F(P) = w_1 f_{length}(P) + w_2 f_{nodes}(P) + \frac{w_3}{2} f_{smooth}(P) + \frac{w_3}{2} f_{AngleStartGoal}(P), \tag{9}$$

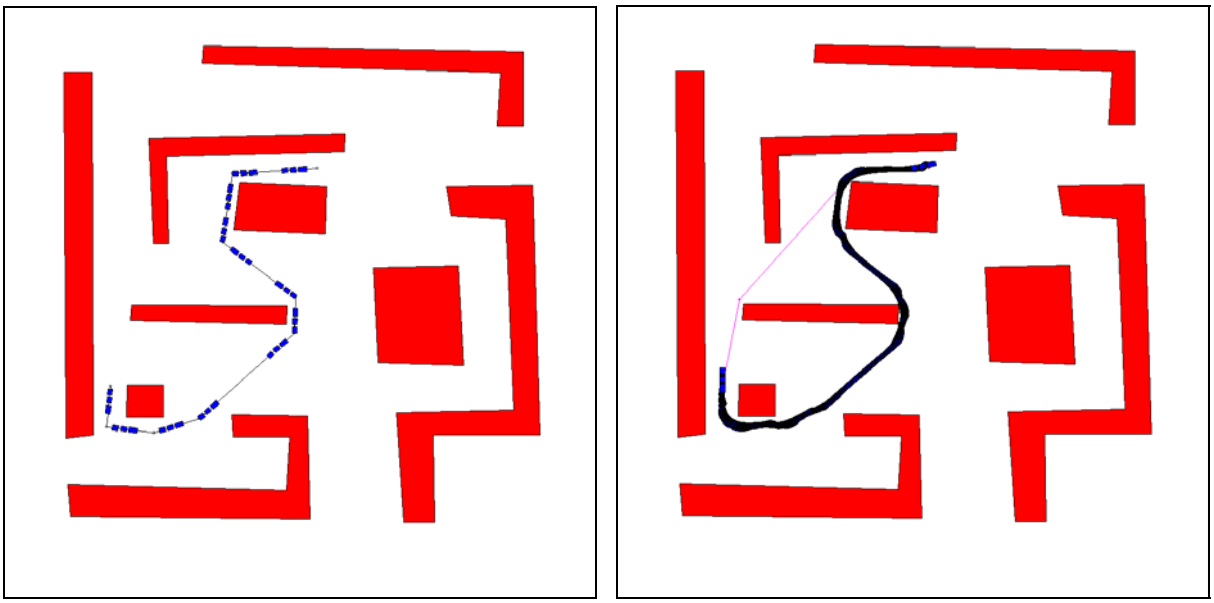
kde $f_{length}(P)$, $f_{nodes}(P)$ a $f_{smooth}(P)$ jsou dány vztahy (4) a pro $f_{AngleStartGoal}(P)$ platí:

$$f_{AngleStartGoal}(P) = \frac{|\beta_1 - \beta_S| + |\beta_n - \beta_G|}{2\pi}. \tag{10}$$

Druhá změna se týká genetických operátorů. V této úpravě již neuvažujeme operátory zkrácení a zlepšení. Naopak zavádíme následující nové operátory:

- Operátor *viditelnosti* nahrazuje operátory zkrácení a zlepšení. Je aplikován na přípustné i nepřípustné chromozomy. Tento operátor vyhlazuje cestu nalezením vhodné cesty grafem viditelnosti, jehož vrcholy jsou náhodně rozmístěny kolem všech původních bodů cesty. Je hledána pouze taková cesta, která splňuje požadavky na hladkost cesty i kritéria pro natočení hran sousedících se startem a cílem. Prohledávání grafu je odvozeno z upraveného Dijkstrova algoritmu navrženého v práci (Krček & Dvořák 2006).
- Operátor *úhlu startu* se snaží zlepšit natočení první hrany vůči natočení robotu ve startu náhodnou malou změnou druhého genu chromozomu. Změna genu nastane pouze pokud není výrazně zhoršena hladkost cesty vůči třetímu genu. Operátor *startu* zachovává přípustnost chromozomu.
- Operátor *úhlu cíle* se snaží zlepšit natočení poslední hrany vůči natočení robotu v cílové pozici. Princip je stejný jako u operátoru startu.

Pro nalezení posloupnosti akcí je možné použít opravný algoritmus (viz podkapitola 3.4). Protože ale transformujeme cestu, která již s neholonomními omezeními počítá (i když jen ve zjednodušené podobě), byla provedena také úprava algoritmu transformace cesty, díky níž byla snížena doba výpočtu transformace.



(a) (b)

Obr. 1. *Cesta pro robot typu auto s dvěma přívěsy nalezená upraveným GA (a) a po provedené transformaci (b). Růžovou barvou je vyznačena cesta nalezená GA z podkapitoly 3.3.*

Nejprve je nutné dopočítat konfigurace podle nalezené cesty genetickým algoritmem. Z každé hrany kratší než $3d_{SubGoal}$, kde $d_{SubGoal}$ je vhodně zvolená konstanta, obdržíme jednu konfiguraci. Poloha robotu v této konfiguraci by měla být umístěna ve střední části hrany a natočení robotu by mělo být rovno natočení hrany. Z každé delší hrany dopočteme dvě konfigurace. Polohy robotu v těchto konfiguracích by měly být umístěny ve vzdálenosti $d_{SubGoal}$ od koncových bodů hrany, natočení robotu je opět rovno natočení hrany. Získané konfigurace (viz obr. 1a) spolu s konfiguracemi startu a cíle tvoří seznam konfigurací *P*.

Cestu respektující neholonomní omezení hledáme obousměrným A* algoritmem. Tento algoritmus konstruuje dva stromy. Uzly v těchto stromech jsou reprezentovány následujícími strukturami: (konfigurace, ohodnocení konfigurace, rodičovská konfigurace, akce kterou byla konfigurace dosažena z rodičovské konfigurace). Uzly prvního stromu jsou ohodnoceny následující funkcí:

$$f_1(Uzel) = g(Start, Uzel) + d(Uzel, Cíl), \qquad (11)$$

kde *g*(*Start, Uzel*) je součet vzdáleností mezi konfiguracemi na cestě ze startu do daného uzlu, *d*(*Uzel, Cíl*) je vzdálenost mezi konfigurací příslušející danému uzlu a cílovou konfigurací. Vzdálenost mezi konfiguracemi je nutno určit individuálně podle významu jednotlivých složek konfigurací. Uzly druhého stromu jsou ohodnoceny následující funkcí:

$$f_2(Uzel) = g(Cíl, Uzel) + d(Uzel, Start), \qquad (12)$$

kde *g*(*Cíl, Uzel*) je součet vzdáleností mezi konfiguracemi na cestě z cíle do daného uzlu, *d*(*Uzel, Start*) je vzdálenost mezi konfigurací příslušející danému uzlu a startovní konfiguraci.

Protože algoritmus pracuje se dvěma stromy, je nutné také rozšířit počet pracovních seznamů. Seznamy *OPEN1* a *OPEN2* obsahují dosud neexpandované uzly a jsou implementovány jako haldy. Seznamy *OCCUP1* a *OCCUP2* slouží k rozpoznávání podobnosti uzlů a současně k testování setkání stromů. Pro rychlé vyhledávání v těchto seznamech jsou tyto seznamy rozděleny do několika dalších, které jsou uspořádány v mřížce a jsou tedy indexovány dvourozměrnými souřadnicemi. Při vkládání prvků do těchto seznamů se nejprve podle složek *x* a *y* konfigurace vkládaného prvku určí odpovídající buňka a tím tedy seznam pro vložení. Při hledání podobné konfigurace v těchto seznamech se neprochází seznamy všechny, ale jen seznam odpovídající konfiguraci, jejíž podobné konfigurace hledáme a seznamy s tímto seznamem sousedící.

Algoritmus transformace cesty může být popsán následujícími kroky:

1. Ze seznamu *P* vyjmeme první prvek a vložíme ho do *S*.
2. Pokud *S* je podobné cílové konfiguraci, pak je algoritmus ukončen, cesta je nalezena.
3. Ze seznamu *P* vyjmeme první prvek a vložíme ho do *G*.
4. Vyprázdníme seznamy *OPEN1*, *OPEN2*, *OCCUP1* a *OCCUP2*.
5. Uzel S vložíme do seznamu *OPEN1* a do seznamu *OCCUP1*.
6. Uzel G vložíme do seznamu *OPEN2* a do seznamu *OCCUP2*.
7. Pokud je seznam *OPEN1* prázdný, algoritmus ukončíme (cesta nebyla nalezena), jinak z tohoto seznamu vyjmeme konfiguraci s minimálním ohodnocením a vložíme ji do *C*.
8. Pokud je natočení konfigurace *S* blízké konfiguraci *G* ($|\beta_S - \beta_G| < \beta_{Near}$, pro vhodně zvolené β_{Near}), pak do množiny U_{neigh} vložíme akci příslušnou uzlu *C* a z množiny *U* vložíme takové akce, které způsobí jen malou změnu směru oproti použití akce uzlu *C*. Jinak položíme $U_{neigh} = U$.
9. Expandujeme uzel *C*, tj. opakujeme následující kroky pro akce $u \in U_{neigh}$:
   a. Aplikujeme akci *u* na uzel *C*.
   b. Pokud obdržíme nekolizní konfiguraci *NC* a seznam *OCCUP1* neobsahuje podobnou konfiguraci, vypočteme $f_1(NC)$ a vložíme *NC* do seznamů *OPEN1* a *OCCUP1*. Jinak pokračujeme aplikací další akce v kroku 9a.
   c. Pokud seznam *OCCUP2* obsahuje konfiguraci *PC* podobnou konfiguraci *NC*, potom zjistíme akce na cestě mezi uzly *PC* a *G*, tyto akce převedeme na inverzní a postupně je aplikujeme na uzel *NC*. Pokud takto obdržíme konfiguraci podobnou *G*, cesta mezi *S* a *G* je nalezena a po vložení *G* do *S* pokračujeme krokem 2.
10. Pokud je seznam *OPEN2* prázdný, pokračujeme krokem 7, jinak z tohoto seznamu vyjmeme konfiguraci s minimálním ohodnocením a vložíme ji do proměnné *C*.
11. Pokud je natočení konfigurace *S* blízké konfiguraci *G*, pak do množiny U_{neigh} vložíme akci příslušnou uzlu *C* a z množiny *U* vložíme takové inverzní akce,

které způsobí jen malou změnu směru oproti použití akce uzlu *C*. Jinak invertujeme všechny akce takto: $U_{neigh} = inv(U)$.

12. Expandujeme uzel *C*, tj. opakujeme následující kroky pro akce $u \in U_{neigh}$:
    a. Aplikujeme akci *u* na uzel *C*.
    b. Pokud obdržíme nekolizní konfiguraci *NC* a seznam *OCCUP2* neobsahuje podobnou konfiguraci, vypočteme $f_2(NC)$ a vložíme *NC* do seznamů *OPEN2* a *OCCUP2*. Jinak pokračujeme aplikací další akce v kroku 12a.
    c. Pokud seznam *OCCUP2* obsahuje konfiguraci *PC* podobnou konfiguraci *NC*, potom zjistíme akce na cestě mezi uzly *NC* a *G*, tyto akce převedeme na inverzní a postupně je aplikujeme na uzel *PC*. Pokud takto obdržíme konfiguraci podobnou *G*, cesta mezi *S* a *G* je nalezena a po vložení *G* do *S* pokračujeme krokem 2.
13. Pokračujeme krokem 7.

V algoritmu může nastat stav, kdy není možné z nového startu dále rozšiřovat nový strom z důvodu zablokování robotu blízkou překážkou. Některá z blízce předcházejících konfigurací byla podobná cíli minulé dvojice stromů, kde však ještě nebylo jasné, že se jedná o konfiguraci v nevhodné větvi stromu. Tato situace nastává s malou pravděpodobností, a to v případech nastavení krátké obchůzkové trasy (viz opravný operátor v podkapitole 3.3) anebo při nastavení malé podobnosti podcílů. Protože není možné zablokovanou konfiguraci dále expandovat, tento problém se projeví předčasným vyprázdněním seznamu *OPEN1*.

Problém je možné řešit tak, že v případě prázdného seznamu *OPEN1* vložíme v kroku 7 do proměnné *S* konfiguraci, která je v dosud nalezené cestě umístěna *n* konfigurací před konfigurací zablokovanou a pokračujeme krokem 4. Toto v kroku 7 opakujeme vždy, když je seznam *OPEN1* prázdný. Experimenty (pro robot na obr. 1) však ukazují, že není potřeba více než pět opakování (pro $n = 5$).

3.6 OPTIMALIZACE PARAMETRŮ

Jedním z cílů práce bylo ověření možnosti nastavování parametrů navržené metody další metodou strojového učení. Z tohoto důvodu byl navržen a otestován genetický algoritmus (GA1), jehož výsledkem jsou hodnoty parametrů genetických algoritmů (GA2) navržených v podkapitolách 3.3 a 3.5 pro konkrétní mapu a pro dané váhy fitness funkcí.

Chromozom má velikost deset genů, přičemž první až devátý gen odpovídá postupně pravděpodobnostem operátorů křížení, velké mutace, malé mutace, opravy, výměny, vyhlazení, zkrácení (úhlu startu a cíle), odstranění a zlepšení (viditelnosti). Desátý gen odpovídá počtu párů chromozomů při selekci. První až devátý gen může nabývat reálných hodnot z intervalu $\langle 0; 1 \rangle$, hodnota desátého genu je celočíselná a nesmí být větší než polovina z počtu chromozomů v populaci optimalizovaného GA2. Počáteční populace je vytvořena zcela náhodně.

Pro každý chromozom GA1 je spuštěn GA2 pro danou testovací množinu startů a cílů. Optimalizujeme-li kvalitu řešení hledaného pomocí GA2, potom je fitness

hodnota chromozomu GA1 rovna průměrné hodnotě fitness nejlepších nalezených cest. Pokud optimalizujeme dobu výpočtu GA2, potom je fitness hodnota chromozomu GA1 rovna průměrné době výpočtu nejlepších nalezených cest pomocí GA2. Je zřejmé, že je pomocí vhodných normalizovaných vah možné současně optimalizovat kvalitu hledané cesty i dobu výpočtu.

GA1 používá binární turnajovou selekci a jednobodové křížení. Operátor mutace provádí náhodné vygenerování nové hodnoty náhodně vybraného genu v chromozomu. Pomocí křížení a mutace je vygenerována nová populace vždy o stejném počtu chromozomů jako populace stará. Ukončení výpočtu GA1 je provedeno tehdy, když již dochází pouze k malým změnám průměrné (nebo nejlepší) hodnoty fitness v populaci.

3.7 KOMBINACE PŘÍPADOVÉHO GRAFU A GA

V této podkapitole je navržen systém případového usuzování, v jehož případové bázi jsou udržovány cesty pro neholonomní mobilní robot. Jako pomocná metoda případového usuzování je použit genetický algoritmus, představený v podkapitole 3.5. Podobně jako v práci (Krček, Dvořák & Hodál 2005), případová báze je implementována jako případový graf. Případový graf u této metody má ovšem odlišnou strukturu, neboť musí uvažovat neholonomní omezení.

Cesta je po absolvování robotem do báze případů vložena jako několik samostatných lineárních segmentů (případů). Každý případ je reprezentován dvojicí opačně orientovaných hran případového grafu $G = (V, E)$, kde V je množina uzlů, $V = \{0,\ 1, 2, \ldots, n\}$ a E je množina orientovaných hran $E \subseteq \{(i, j) \mid i, j \in V\}$. K vložení nového případu dojde pouze tehdy, pokud báze případů neobsahuje již případ podobný vkládanému případu. Případ, jehož jedna z dvojice hran je (u, v), je podobný případu, jehož jedna z hran je (i, j), pokud $MB((u, v), (i, j)) < d_{proximity}$, kde $d_{proximity}$ je maximální hodnota míry blízkosti a $MB((u, v), (i, j))$ je míra blízkosti daná vztahem

$$MB\big((u,v),(i,j)\big) = \min\left\{\begin{array}{l}\max\{d(i,u), \min\{vz(j,u,v), vz(v,i,j)\}\},\\ \max\{d(i,v), \min\{vz(j,u,v), vz(u,i,j)\}\},\\ \max\{d(j,u), \min\{vz(i,u,v), vz(v,i,j)\}\},\\ \max\{d(j,v), \min\{vz(i,u,v), vz(u,i,j)\}\}\end{array}\right\}, \qquad (13)$$

kde $d(i, j)$ je euklidovská vzdálenost uzlů i a j a $vz(v,i,j)$ je vzdálenost bodu v od hrany (i, j). Pomocí převrácené hodnoty míry blízkosti je možné odvodit míru podobnosti.

Hrany grafu mohou mít společné své krajní body. Pokud přidávaný případ protíná již existující hranu grafu, pak jsou cesta i hrana rozděleny do menších případů pouze tehdy, pokud rozdělením nevznikne případ o délce kratší, než je minimální stanovená délka. Je tedy povoleno křížení hran, aniž by musel být v jejich průsečíku uzel.

Případ (dvojice opačně orientovaných hran) je v bázi případů uložen se strukturou $(t_{Sum}, t_{SumW}, n_1, n_2, n_3)$, kde $t_{Sum} = \sum t$ je součet normálních dob průchodu případem, $t_{SumW} = \sum t_W$ je součet dob čekání t_W při průchodu případem, n_1 je celkový počet průchodů případem, n_2 je počet průchodů případem s čekáním $t_W \le t_{MaxW}$ a n_3 je počet případů, kdy byla použita jiná cesta ($t_W > t_{MaxW}$). Ohodnocení případu t_E je dáno následujícím vztahem:

$$t_E = \frac{t_{Sum}}{n} + \frac{n_2}{n}\left(\frac{t_{Sum}}{n_1} + \frac{t_{SumW}}{n_2}\right) + \frac{n_3}{n}\left(t_{MaxW} + M\right), \qquad (14)$$

kde $n = \sum_{i=1}^{3} n_i$, t_{MaxW} je maximální doba čekání a M je vhodně volená penalizace neprůchodné hrany (např. nejdelší cesta ze startu do cíle se započtením čekacích dob).

V důsledku kinematických omezení robotu nelze pro prohledávání grafu použít běžných metod prohledávání. Dále popsaný algoritmus hledání cesty rozšiřuje algoritmus, který je navržený v práci (Krček & Dvořák 2006) o fiktivní uzel g', díky kterému můžeme nyní uvažovat i cílové natočení robotu.

Symbolem $d_s(j, k)$ je označena doba průchodu z uzlu s do uzlu k přes hranu (j, k) v grafu $G = (V, E)$. Symbol $p_s(i, j)$ označuje počáteční uzel hrany bezprostředně předcházející hraně (i, j) na nejkratší cestě z uzlu s do uzlu j přes hranu (i, j). Symbol $\beta(i, j)$ odpovídá natočení hrany (i, j).

Rozšířený algoritmus hledání přípustné cesty z uzlu s do uzlu g lze popsat následujícími kroky:

1. Do V dočasně přidáme fiktivní uzly s' a g' tak, aby se natočení fiktivních hran (s', s) a (g, g') rovnalo natočení robotu ve startu $\beta(s', s) = \beta_S$ a cíli $\beta(g, g') = \beta_G$.
2. Do V dočasně přidáme m uzlů náhodně vygenerovaných v okolí startu i cíle a nekolidujících s překážkami..
3. Do E dočasně přidáme takové nekolizní hrany (i, j), které jsou incidentní alespoň s jedním dočasným uzlem a současně splňují podmínku délky $d(i, j) \ge d_{Min} \wedge d(i, j) \le d_{Max}$.
4. Do E dočasně přidáme nekolizní hrany (i, j) spojující ostatní uzly takové, které nekříží žádnou z již existujících hran v E a současně splňují podmínku délky $d(i, j) \ge d_{Min} \wedge d(i, j) \le d_{Max}$.
5. Pro hranu (s', s) položme ohodnocení $d_s(s', s) = 0$ a pro ostatní hrany položme $d_s(i, j) = \infty$.
6. Do haldy H vložíme všechny hrany z E.
7. Z haldy H vyjmeme hranu s minimálním ohodnocením a označíme jako (k, r).
8. Pokud $(k, r) = (g, g')$, algoritmus ukončíme, cesta je nalezena.

9. Pokud je ohodnocení $d_s(k, r) = \infty$, pak algoritmus ukončíme, cestu grafem G není možné vyhledat.
10. Prozkoumáme všechny hrany $(r, i) \in E$, pro něž platí $d(r,i) \geq d_{Min} \wedge d(r,i) \leq d_{Max}$. Pokud $(k,r) = (s',s)$, pak musí též platit $|\beta(k,r) - \beta(r,i)| \leq \beta_{StartMax}$, pokud $(r,i) = (g,g')$, pak musí také platit $|\beta(k,r) - \beta(r,i)| \leq \beta_{GoalMax}$, jinak musí platit $|\beta(k,r) - \beta(r,i)| \leq \beta_{SegMax}$. Když hrana (r, i) splňuje tyto podmínky a $d_s(r, i) > d_s(k, r) + t_E$, pak položíme $d_s(r, i) = d_s(k, r) + t_E$ a $p_s(r, i) = k$. Ohodnocení t_E je vypočteno podle vztahu (14) pro hranu (r, i). Pokud je (r, i) dočasná hrana, pak je t_E odhad doby průchodu touto hranou.
11. Návrat na krok 7.

Cesta robotu z uzlu s do uzlu g bude určena jako posloupnost hran uložených v zásobníku Z, přičemž počáteční hrana se bude nacházet na vrcholu zásobníku. Určení nejkratší cesty z s do g lze popsat takto:

1. Položíme $Z = \varnothing$. Položíme $(k,r) = (g,g')$.
2. Do zásobníku Z vložíme hranu (k, r).
3. Pokud je $(k,r) = (s',s)$ pak ze zásobníku vyjmeme fiktivní hrany (hrany na vrcholu a dně zásobníku), z grafu G odstraníme dočasné hrany i uzly a algoritmus ukončíme, cesta je uložena v zásobníku Z.
4. Položme $(k, r) = (p_s(k, r), k)$ a pokračujeme krokem 2.

Pokud v případovém grafu G nebyla cesta nalezena nebo selhala-li následná transformace této cesty, pak cestu hledáme pomocí GA. Cestu nalezenou GA se pokusíme upravit podle případového grafu tak, že pokud leží některý z vnitřních uzlů nalezené cesty v okolí uzlů množiny V, pak tento vnitřní uzel cesty nahradíme nejbližším uzlem z množiny V. Tímto dosáhneme toho, že po absolvování cesty robotem bude v případovém grafu aktualizována již existující hrana, případně bude vytvořena nová hrana mezi existujícími uzly. Pokud touto úpravou vznikne nepřípustná cesta nebo selže-li transformace cesty, je nutné použít cestu nalezenou GA.

Po přidání cesty nebo po aktualizaci její struktury $(t_{Sum}, t_{SumW}, n_1, n_2, n_3)$ je vhodné odstranit staré a málo používané případy. Tímto dosáhneme možného přidání nových hran, které mohly být těmito starými málo používanými avšak podobnými případy blokovány. Po dosažení maximální velikosti případové báze je možné spustit odstranění starých případů samostatně. Tímto sice můžeme přijít na nějakou dobu o kvalitní případy, ale současně se naskýtá možnost najít v prostředí s dynamickými změnami případy lepší.

4 IMPLEMENTACE

Pro ověření funkce a experimentování s navrženými metodami bylo v průběhu vývoje vytvořeno několik simulačních programů. K tvorbě programů bylo použito

vývojové prostředí Borland Delphi 7.0. Finální simulační aplikace se jmenuje *Robot2010*. V této aplikaci jsou implementovány metody z podkapitol 3.3 až 3.7, které lze považovat za nejvíce přínosné.

Navržené metody jsou implementovány tak, aby nebyly na simulační aplikaci *Robot2010* závislé a byly jednoduše použitelné i mimo tuto aplikaci. Podobně není navržený systém vázán na jeden kinematický model robotu. Díky použitému objektovému přístupu je možné další modely robotů do systému snadno implementovat.

5 EXPERIMENTÁLNÍ OVĚŘENÍ

5.1 EXPERIMENTY NA MŘÍŽCE

Z navržených metod pro mřížku dosahují nejlepších výsledků metoda kombinující A* s normálním případovým grafem a metoda kombinující A* s „jemnou strukturou“ případového grafu (Dvořák & Krček 2005). Z experimentů plyne, že po naplnění báze případů pracují obě navržené metody rychleji než algoritmus A* pro všechny testované velikosti okolí.

5.2 KOMBINACE PŘÍPADOVÉHO GRAFU A RRT

Provedené experimenty naznačují, že použití případového grafu může přinést významnou úsporu ve srovnání se samotným algoritmem RRT. Nevýhodou navrženého přístupu je vysoká vazba na model robotu, ale bohužel i řešení podsystémů doplňování nových případů a údržby báze případů v aktuálním stavu, které se během vývoje ukázalo být příliš komplikované. Za určitou nevýhodu může být považováno také to, že tato metoda neuvažuje natočení robotu v cílové pozici.

5.3 ZNOVUPOUŽITÍ POPULACE GA

V těchto experimentech bylo zkoumáno chování GA navrženého v podkapitole 3.3. Parametry GA byly nastaveny experimentálně. Experimenty probíhaly ve scéně podobné bludišti. V prvním experimentu byla zkoumána možnost použití části naučené populace předchozího řešení jako část počáteční populace nového problému, který je charakterizován novým startem a/nebo novým cílem ve stejné scéně. Ve druhém experimentu byla zkoumána možnost použití konečné populace současného řešení pro přeplánování cesty z důvodu výskytu neznámé překážky. V tomto případě přeplánování začíná v novém startu, ale cíl zůstává stejný. V obou experimentech došlo použitím chromozomů z naučené populace ke zrychlení výpočtu bez zhoršení hodnoty fitness funkce.

5.4 OPTIMALIZACE PARAMETRŮ GA

V těchto experimentech byla provedena optimalizace parametrů GA navržených v podkapitole 3.3 a 3.5. Princip optimalizace je vysvětlen v podkapitole 3.6.

V případě GA pro holonomní robot byla prováděna pouze optimalizace kvality hledané cesty, protože doba výpočtu byla stále v přípustných mezích. V případě GA pro neholonomní robot bylo nutné optimalizovat kromě kvality hledané cesty i dobu výpočtu. Ze zlepšování hodnot fitness funkce v průběhu výpočtu je zřejmé, že navržený způsob optimalizace je účelný. Ve všech následujících experimentech byly použity vypočtené optimalizované hodnoty parametrů.

5.5 POROVNÁNÍ GA A GV

V těchto experimentech bylo provedeno porovnání genetického algoritmu pro holonomní robot navržený v podkapitole 3.3 s algoritmem používajícím graf viditelnosti (GV, viz podkapitola 2.1). Experimenty probíhaly ve scéně, v níž byly náhodně generovány obdélníkové překážky. Cesty nalezené pomocí GV byly ohodnoceny fitness funkcí (3). Z porovnání průměrných hodnot fitness plyne, že GA je schopné GV konkurovat. GA poskytuje mírně lepší řešení než GV. Je to dáno tím, že v GV je hledána pouze nejkratší cesta a oproti GA není tato cesta vyhlazována. Předpokládáme-li opakované generování GV, pak při použití GA můžeme (ve scéně jejíž překážky obsahují přibližně 400 vrcholů) získat až šestinásobné zrychlení výpočtu.

5.6 POROVNÁNÍ METOD PRO NEHOLONOMNÍ ROBOTY

V následujících experimentech je ověřena funkčnost a použitelnost metody genetického algoritmu s následnou transformací, navržené v podkapitole 3.5. Tato metoda je implementována jako jednovláknová (GA-TR). Pro spuštění metody ve víceprocesorovém (či vícejádrovém) systému je tato metoda také implementována pro paralelní zpracování jako vícevláknová aplikace (GA-TR-par).

Experimenty byly prováděny ve scéně 500×500 se třemi různými modely o různých složitostech (LaValle 2006): robot s diferenciálním řízením, robot typu „automobil“ a souprava robotu a dvou přívěsů pohybující se pouze vpřed.

Nalezené cesty byly hodnoceny podle tří kritérií. Prvním kriteriem je plynulost průchodu cestou, druhým je délka cesty a třetím je doba výpočtu plánování cesty. Pokud během testování metoda GA-TR či GA-TR-par nenalezla cestu, byla spuštěna opakovaně. Výsledná doba výpočtu zahrnuje všechny pokusy hledání.

V experimentech, které byly prováděny pro robot s diferenciálním řízením, bylo provedeno srovnání GA-TR (GA-TR-par) se stávajícími metodami RRT ve variantách RRT-Z (vylepšení RRT-GoalZoom, LaValle 2001) a RRT-R (vylepšení RRT-GoalRegionBiasedConf, Qu 1999). Tyto metody jsou pro tento jednoduchý model rychlé, avšak často u těchto metod docházelo k uváznutí v lokálním minimu. Dále bylo pro tento jednoduchý model provedeno srovnání s obousměrným A* algoritmem. Do stanovené maximální doby výpočtu tento algoritmus cestu nenalezl v 23,2% případů. Vícevláknová verze (A*-par) selhala v 11,6% případů.

V experimentech prováděných pro robot typu „automobil“ selhává obousměrný algoritmus A* v 68,2% případů, verze A*-par selhává v 32,4% případů. Algoritmy

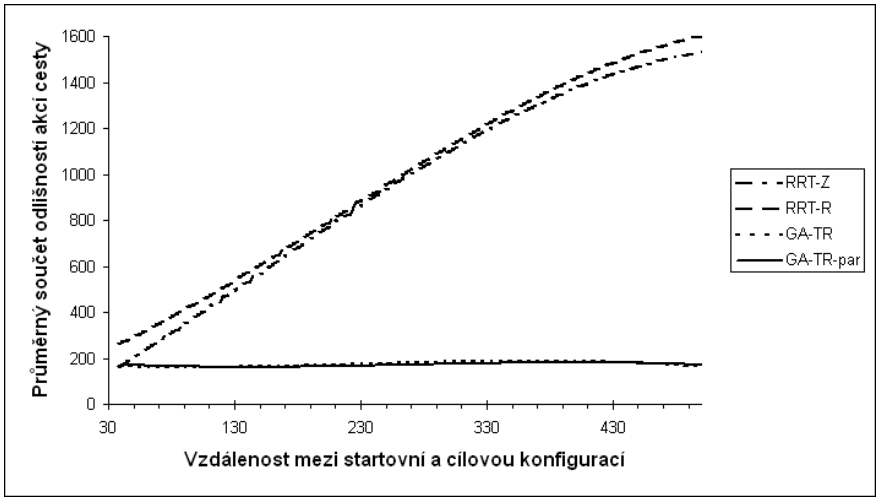
A* a A*-par tedy nebyly do srovnání vůbec zařazeny a bylo provedeno pouze porovnání metod GA-TR, GA-TR-par, RRT-Z a RRT-R.

U experimentů pro soupravu robotu s přívěsy se projevuje neschopnost metod RRT pracovat se složitějšími modely, které nejsou schopny zpětného pohybu. I když byla pro RRT-Z a RRT-R nastavena trojnásobně větší tolerance cíle, metoda RRT-Z selhává v 45,3% případů a RRT-R selhává v 49,5% případů. Obousměrný A* zde selhává v 68,4% případů. Pro robot s přívěsy bylo tedy provedeno pouze srovnání navržených metod GA-TR a GA-TR-par.

5.6.1 Porovnání plynulosti průchodů

Plynulost průchodu cestou *P* je závislá na hodnotě součtu odlišností vždy po sobě následujících akcí cesty *P*. Čím je součet odlišností mezi akcemi větší, tím bude průchod cestou méně plynulý.

Z výsledků těchto experimentů vyplývá, že nejlepší průchod poskytují cesty nalezené metodou GA-TR (nebo její paralelní verzí). Největší rozdíl je patrný u modelu robotu typu automobil (obr. 2), kde je při nejdelších vzdálenostech mezi konfiguracemi startu a cíle dosaženo až sedmkrát lepší průchodnosti než poskytují metody RRT. U robotu s diferenciálním řízením dosahuje lepší průchodnosti algoritmus A*, avšak tento algoritmus v některých případech selhal (viz předcházející podkapitola).

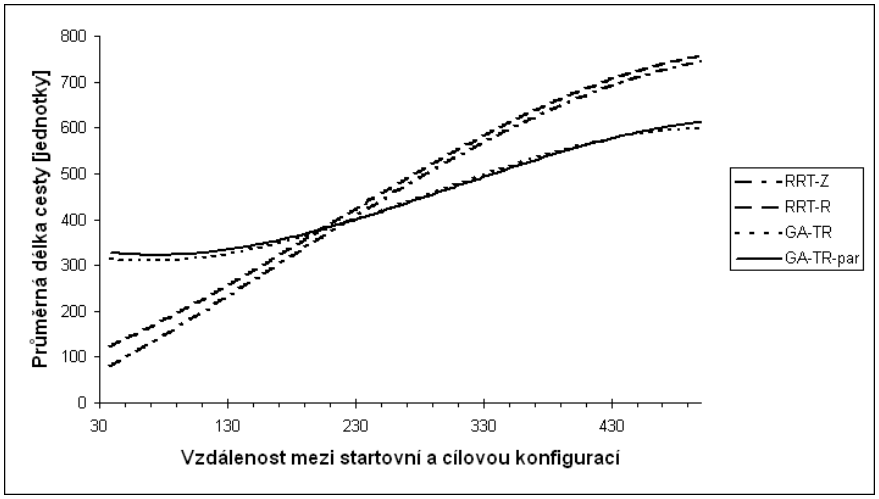


Obr. 2. *Závislosti průměrného součtu odlišností akcí nalezené cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu typu „automobil“.*

5.6.2 Porovnání délek nalezených cest

Délka nalezené cesty odpovídá délce křivky, kterou zanechá referenční bod robotu po průchodu cestou. Pro robot s diferenciálním řízení a vzdálenosti konfigurací menší než 130 jednotek, poskytují nejkratší cesty metody RRT. Pro robotu typu „automobil“ podává RRT kratší cesty do vzdálenosti přibližně 200 jednotek (obr. 3). Uvážíme-li však průchodnost cesty (viz předchozí podkapitola), pak můžeme tvrdit, že kromě robotu s diferenciálním řízením a vzdálenosti

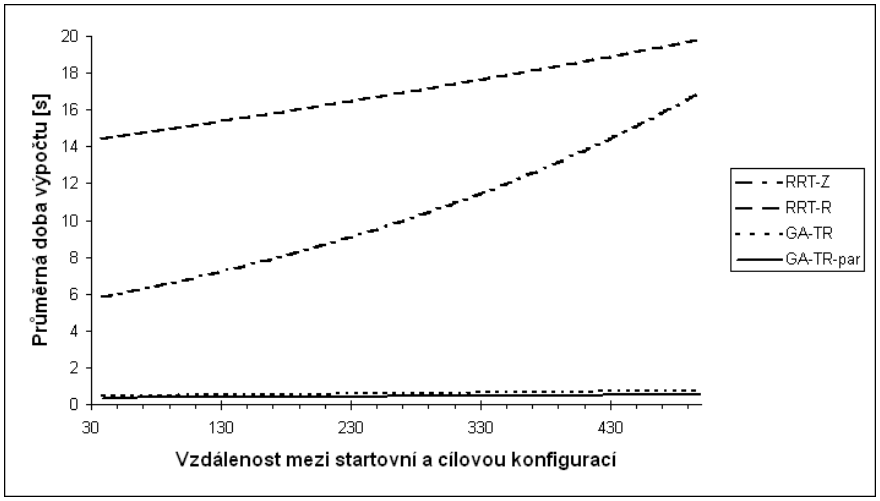
konfigurací menší než 80 jednotek, poskytuje nejlepší řešení metoda GA-TR nebo její paralelní verze GA-TR-par. Vzrůstající délka cesty je způsobena vzrůstající vzdáleností mezi konfiguracemi startu a cíle.



Obr. 3. *Závislosti průměrné délky nalezené cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu typu „automobil“.*

5.6.3 Porovnání dob výpočtu

Pro každé hledání cesty byla měřena doba výpočtu. Navržená metoda GA-TR (případně GA-TR-par) poskytuje nejrychlejší výpočet pro všechny typy robotů. V případě robotu s přívěsy, pro které je generování stromů při transformaci již časově náročné, lze pozorovat výhody paralelního zpracování, které zde snižuje čas výpočtu téměř o polovinu.



Obr. 4. *Závislosti průměrné doby výpočtu hledání cesty na vzdálenosti mezi startovní a cílovou konfigurací robotu typu „automobil“.*

5.7 EXPERIMENTY S UČENÍM PŘÍPADOVÉHO GRAFU

Pro testování systému případového usuzování v kombinaci s genetickým algoritmem (viz podkapitola 3.7) byl vybrán nejsložitější model robotu s přívěsy.

Tento model požaduje oproti jednodušším modelům dosti vysoké nastavení hodnot $\beta_{StartMax}$, $\beta_{GoalMax}$, β_{SegMax}, d_{Min} a $d_{SubGoal}$ a klade tak na vyhledávání cesty v případovém grafu největší omezení. Pro jednodušší modely podává tento navržený systém lepší výsledky. Experimenty probíhaly ve scéně 1000×1000 jednotek (viz obr. 5). Experimenty začínaly vždy s prázdnou případovou bází a k učení případů tak docházelo v průběhu experimentu.

Obr. 5. *Příklad případového grafu a cesty pro robot s přívěsy ve scéně o velikosti 1000×1000 s neznámou překážkou.*

Po každém přidání (či aktualizaci) případu do případové báze bylo provedeno odstranění případů starších dvaceti hledání, pro které platí $n_1 = 1$ a $n_2 = 0$. V experimentech bylo předpokládáno, že $t_{MaxW} = \infty$. Odstraňování případů při překročení maximální velikosti případové báze nebylo uvažováno, neboť zmíněné odstraňování starých případů zvládalo v průběhu experimentů udržovat velikost případové báze v přípustných mezích (do 300 případů).

5.7.1 Porovnání s metodou GA-TR-par

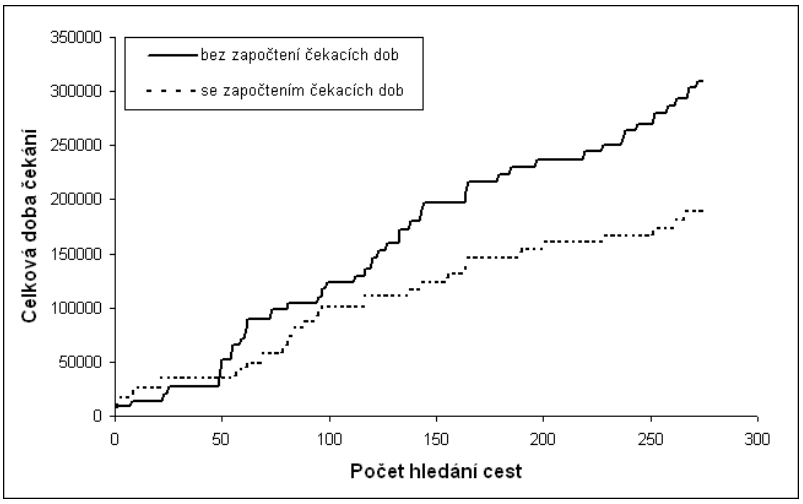
V těchto experimentech byly pro každý pár náhodně vygenerovaných konfigurací hledány cesty metodou GA-TR-par a pomocí systému případového usuzování s následnou transformací v paralelní verzi (CG-TR-par). Stejně jako v podkapitole 5.6, byly nalezené cesty hodnoceny podle plynulosti průchodu cestou, délky cesty a doby výpočtu metody. Experimenty byly prováděny pro různé velikosti okolí *o* a počty generovaných bodů v tomto okolí *m*.

Na základě těchto experimentů je možné se domnívat, že použití navržené metody CG-TR-par nepřináší ve statických scénách významné zlepšení plánování cesty. Pokud je ovšem případový graf udržován v aktuálním stavu a tedy se v něm odráží dynamické změny prostředí, pak je použití metody CG-TR-par již přínosné (viz následující podkapitola).

5.7.2 Plánování cesty v částečně známém prostředí

Pro experimenty v této podkapitole byla do scény vložena neznámá překážka o velikosti přibližně 50×50 jednotek. Pravděpodobnost výskytu této překážky byla stanovena na hodnotu 0,8. Minimální doba výskytu této překážky byla stanovena na 5000 časových jednotek a maximální doba výskytu na 10000 časových jednotek. Pro stejnou testovací množinu konfigurací jako v předcházející podkapitole bylo prováděno hledání pomocí CG-TR-par dvakrát. V první sérii testů bylo uvažováno ohodnocení případů podle navrženého vztahu (14), v druhé sérii testů nebyla při přidávání (aktualizaci) případů uvažována doba čekání t_W.

Z experimentů plyne, že při vhodném nastavení přináší použití metody CG-TR-par zřetelné snížení celkové doby průchodu robotu prostředím s neznámou překážkou (viz obr. 6). Jako vhodná velikost okolí se jeví *o=100* při počtu generovaných bodů *m=40*.



Obr. 6. *Celková doba čekání v závislosti na počtu hledaných cest pro o=100 a m=40*

6 ZÁVĚR

Tato disertační práce zkoumá použití metod strojového učení pro plánování cesty autonomního lokomočního robotu. V souvislosti s plánováním cesty se z metod strojového učení uplatňují především případové usuzování, neuronové sítě, posilované učení, rojová inteligence a genetické algoritmy.

Prvá část práce je přehledem řady stávajících metod pro plánování cesty holonomních i neholonomních robotů. Ve druhé části práce je navržena řada originálních metod pro plánování cesty holonomního i neholonomního robotu, které jsou založeny především na případovém usuzování a genetických algoritmech. Třetí část práce je věnována popisu implementace těchto metod a čtvrtá část rozebírá výsledky experimentů s navrženými metodami.

Návrhy metod pro plánování cesty na mřížce jsou výsledkem společného výzkumu vedoucího práce a autora práce. Základní principy algoritmů těchto metod

jsou dílem především vedoucího práce a veškeré implementace a experimentální ověření provedl autor práce. Návrhy metod pro plánování cesty ve spojitém prostoru jsou kromě základního návrhu genetického algoritmu s některými specifickými operátory a matematického popisu upraveného Dijkstrova algoritmu již původní prací autora, stejně jako implementace a experimentální ověření těchto metod.

Experimentální ověření bylo provedeno postupně se všemi navrženými metodami. V oblasti plánování cesty na mřížce byly srovnány dvě stávající metody plánování cesty holonomních robotů s dvěma metodami navrženými. Z těchto experimentů plyne, že po naplnění báze případů pracují navržené metody rychleji než algoritmus A* pro všechny testované velikosti okolí.

Z experimentálního ověření metody kombinující případový graf a pravděpodobnostní stromy vyplývá, že při vhodné volbě okolí a případové bázi dostatečně pokrývající svými případy scénu lze dosáhnout vůči samostatnému používání RRT jak zrychlení doby výpočtu, tak i vyhledání kratších cest.

V dalších experimentech věnovaných navrženému genetickému algoritmu byla zkoumána možnost použití části naučené populace předchozího řešení v počáteční populaci nového problému a možnost použití konečné populace současného řešení pro přeplánování cesty z důvodu výskytu neznámé překážky. Z těchto experimentů je patrné, že pro testovanou scénu a holonomní robot je nejlepších výsledků dosaženo, když celá stará populace je použita jako počáteční populace nového problému.

V práci je také ověřena možnost nastavování parametrů navrženého genetického algoritmu dalším genetickým algoritmem. Toto je provedeno pro obě varianty navrženého algoritmu (pro holonomní i neholonomní robot). Klesající průběh hodnot fitness funkce v průběhu výpočtu dokazuje, že navržený způsob optimalizace je účelný.

V oblasti plánování cesty neholonomních robotů byly srovnávány navržené metody s metodami RRT a s obousměrným A* algoritmem. Experimenty byly prováděny se třemi různými modely neholonomních robotů. Ze srovnání metod plyne, že navržená metoda kombinující genetický algoritmus s následnou transformací cesty poskytuje v mnoha případech kvalitnější cesty a to i při kratší době výpočtu. Pro složitější modely robotů je navržená metoda dokonce jediná použitelná, protože ostatní implementované metody zde často selhávají.

Závěrečné experimenty byly prováděny v simulovaném částečně známém prostředí. Pomocí navrženého systému, který je založen na případovém usuzování, bylo dosaženo zřetelného snížení průměrné doby průchodu robotu prostředím.

Navržený systém plánování cesty neholonomního robotu je implementován tak, aby nebyl vázán na jeden kinematický model robotu. Díky použitému objektovému přístupu je možné další modely robotů do systému snadno implementovat. Podobně není navržený systém vázán na simulační prostředí a je možné jeho použití v reálném robotu.

7 SEZNAM POUŽITÉ LITERATURY

AAMODT, A.; PLAZA, E. 1994. Case-Based Reasoning: Foundational Issues, Methodological Variations, and System Approaches. *AI Communications*, vol. 7, no. 5, pp. 39-59.

BURCHARDT, H.; SALOMON, R. 2006. Implementation of Path Planning Using Genetic Algorithms on Mobile Robots. In *Proceedings of the IEEE World Congress on Computational Intelligence (WCCI 2006), Congress on Evolutionary Computation (CEC 2006)*, Vancouver, Canada, pp. 1831-1836.

DRUMMOND, C. 2002. Accelerating Reinforcement Learning by Composing Solutions of Automatically Identified Subtasks. *Journal of Artificial Intelligence Research*, 16, pp 59-104.

DVOŘÁK, J. & KRČEK, P. 2005. Mobile Robot Path Planning by Means of Case-Based Reasoning. *Engineering Mechanics*, vol. 12, No. A1, pp. 219-226.

DVOŘÁK, J.; KRČEK, P. 2006. Using Case-Based Reasoning and Graph Searching Algorithms for Mobile Robot Path Planning. In *Proceedings of the 12th International Conference on Soft Computing MENDEL 2006*, Brno, pp. 151-156.

DVOŘÁK, J.; KRČEK, P. 2008. Using Genetic Algorithms for Mobile Robot Path Planning. In *Proceedings of the 14th International Conference on Soft Computing MENDEL 2008*, Brno, pp. 32-37.

DVOŘÁK, J.; KRČEK, P. 2009. Path Planning for Nonholonomic Robot. In *Proceedings of the 15th International Conference on Soft Computing MENDEL 2009*, Brno, pp. 336-343.

FERGUSON, D.; LIKHACHEV, M.; STENTZ, A. 2005. A Guide to Heuristic-based Path Planning, *American Association for Artificial Intelligence*.

HAIGH, K.; SHEWCHUK, J. 1994. Geometric Similarity Metrics for Case-Based Reasoning. *Case-Based Reasoning, The AAAI-94 Workshop*, Seattle, WA, August, AAAI Press, pp. 182-187.

HODÁL, J.; DVOŘÁK, J.; KRČEK, P. 2005. Systém pro plánování cesty robota případovým usuzováním, In *Proceedings of XXVIIth Internatinal Autumn Colloquium Advanced Simulation of Systems (ASIS 2005)*, Přerov, s. 255-260.

KRČEK, P.; DVOŘÁK, J. 2009. Plánování cesty mobilního robotu pomocí genetických algoritmů. *Šedesát Let Kybernetiky*, Akademické nakladatelství CERM, pp. 90-95.

KRČEK, P.; DVOŘÁK, J.; HODÁL, J. 2005. Mobile Robot Path Planning by Means of Case-Graph and Genetic Algorithms. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2005*, Svratka, s. 169-170.

KRČEK, P; DVOŘÁK, J. 2006. Mobile Robot Path Planning by Means of Case-Based Reasoning and Rapidly-Exploring Random Trees. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2006*, Svratka, s. 180-181.

KRČEK, P; DVOŘÁK, J. 2007. Mobile Robot Path Planning by Means of Genetic Agorithms. In *Book of Extended Abstracts of the National Conference with International Participation Engineering Mechanics 2007*, Svratka, s. 133-134.

KREJSA, J.; VĚCHET, S. 2005. Rapidly Exploring Random Trees used for Mobile Robots Path Planning. *Engineering Mechanics*, vol. 12, no. 4. pp. 231-237.

KRUUSMAA, M. 2003. Global Level Path Planning for Mobile Robots in Dynamic Environments. *Journal of Intelligent and Robotic Systems*, 38, pp. 55-83.

LAVALLE, S. M. 2006. *Planning Algorithms*. Cambridge University Press, (http://planning.cs.uiuc.edu/).

LAVALLE, S. M.; KUFFNER, J. J. 2001. Rapidly-Exploring Random Trees: Progress and Prospects. In Donald, B. R., Lynch, K. M.; Rus, D. editors, *Algorithmic and Computational Robotics: New Directions*, A K Peters, Wellesley, MA, pp. 293-308.

LEBEDEV, D.; STEIL, J. J.; RITTER, H. J. 2005. The Dynamic Wave Expansion Neural Network Model for Robot Motion Planning in Time-Varying Environments. *Neural Networks*, 18, pp. 267-285.

LEI, K.; QIU, Y.; HE, Y. 2006. A Novel Path Planning for Mobile Robots Using Modified Particle Swarm Optimizer. In *Systems and Control in Aerospace and Astronautics, 2006. ISSCAA 2006. 1st International Symposium on*, Harbin, pp. 981-984.

MEYER, J.-A.; FILLIAT, D. 2003. Map-Based Navigation in Mobile Robots: II. A Review of Map-Learning and Path-Planning Strategies. *Cognitive Systems Research*, vol. 4, no. 4, pp. 283-317.

PRIYA, T. K.; SRIDHARAN, K. 2006. A parallel Algorithm, Architecture and FPGA Realization for High Speed Determination of the Complete Visibility Graph for Convex Objects. *Microprocessors and Microsystems*, vol. 30, issue 1, pp. 1-14.

QU, J. 1999. Nonholonomic Mobile Robot Motion Planning. *Final Report,* Iowa State University (http://msl.cs.uiuc.edu/~lavalle/cs576_1999/projects/junqu/).

RITTHIPRAVAT, P.; MANEEWARN, T.; LAOWATTANA, D.; NAKAYAMA, K. 2002. Obstacle Avoidance Using Modified Hopfield Network for Multiple Robots. *Computer and Communications,* Phuket, Thailand, vol. 1, pp. 30-31.

ŠEDA, M. 2005. Motion Planning in the Plane with Polygonal Obstacles. *Engineering Mechanics*, vol. 12, no. 4. pp. 253-258.

VIET, N. H.; VIEN, N.A.; LEE, S. G.; CHUNG, T. Ch. 2008. Obstacle Avoidance Path Planning for Mobile Robot Based on Multi Colony Ant Algorithm. In *Proceedings of the First International Conference on Advances in Computer-Human Interaction*, pp. 285-289.

ZHENG, C.; DING, M.; ZHOU, C.; LI, L. 2004. Coevolving and Cooperating Path Planner for Multiple Unmanned Air Vehicles. *Engineering Applications of Artificial Intelligence*, vol. 17, issue 8, pp. 887-896.

AUTOROVO CV

Osobní údaje

Ing. Petr Krček, narozen 9. října 1977 v Brně, ČR, ženatý, dcera Barbora

Pracoviště

Ústav automatizace a informatiky
Fakulta strojního inženýrství
Vysoké učení technické v Brně
Technická 2896/2
616 69 Brno

Funkce na pracovišti

technický pracovník

Vzdělání a akademická kvalifikace

2003: Ing., Fakulta strojního inženýrství VUT v Brně, obor Inženýrská informatika a automatizace
2000: Bc., Fakulta strojního inženýrství VUT v Brně, obor Aplikovaná informatika a řízení
1996: maturita, Střední odborné učiliště spojů Brno, obor Mechanik elektronik se zaměřením pro číslicovou a řídicí techniku

Přehled zaměstnání

2006-dosud: technický pracovník, Ústav automatizace a informatiky FSI VUT v Brně

Pedagogická činnost

Vedení cvičení předmětů: Počítačové sítě, Operační systémy, Expertní systémy, Informatika 1
Vedení diplomových a bakalářských prací

Vědeckovýzkumná činnost

Plánování cesty robotu (grafové algoritmy, genetické algoritmy, případové usuzování)

ABSTRACT

As already clear from the title, this dissertation deals with autonomous locomotive robot path planning, based on machine learning. Robot path planning task is to find a path from initial to target position without collision with obstacles so that the cost of the path is minimized. Autonomous robot is such a machine which is able to perform tasks completely independently even in environments with dynamic changes. Path planning in dynamic partially known environment is a difficult problem. Autonomous robot ability to adapt its behavior to changes in the environment can be ensured by using machine learning methods. In the field of path planning the mostly used methods of machine learning are case-based reasoning, neural networks, reinforcement learning, swarm intelligence and genetic algorithms.

The first part of this thesis introduces the current state of research in the field of path planning. Overview of methods is focused on basic omnidirectional robots and robots with differential constraints.

In the thesis, several methods of path planning for omnidirectional robot and robot with differential constraints are proposed. These methods are mainly based on case-based reasoning and genetic algorithms.

Case-based reasoning (CBR) solves a new problem by adapting known solutions of similar previously solved problems. It seems that the CBR is a suitable method for robot control because robotic applications usually include repeated tasks. Case-based path planning is impossible if not combined with other path planning methods. Using these methods is necessary in situations, where a CBR system begins its work with empty case base, or a retrieved solution is not good enough and must be rejected or adapted. The genetic algorithm is the most used complementary method in this work.

Genetic algorithms (GA) are ranked among evolutionary heuristic methods, which are often used to solve complex optimization problems. Early applications of genetic algorithms to problems of path planning were insufficient due to the use of conventional GA with binary representation of the chromosome of fixed length, and only two basic genetic operators (for crossing and mutation). Subsequent successful applications modified the classical GA by non-binary representation, variable length chromosome, and especially problem-specific genetic operators, coupled with some heuristics.

All proposed methods were implemented in simulation applications. Results of experiments carried out in these applications are part of this work. For each experiment, the results are analyzed. The experiments show that the proposed methods are able to compete with commonly used methods, because they perform better in most cases.